CHAPTER 5

CENTRAL PROCESSING UNITS AND BUSES

INTRODUCTION

Digital computers have three major functional areas: **central processing unit (CPU), memory,** and **input/output (I/O).** This applies whether the computer is an 8-bit microprocessor or a 32-bit mainframe. Two other areas must be considered: the **system buses** and the **power supply.** They, too, play a major role with the functional areas of the computer. The buses are the means by which the CPU, memory, and I/O communicate with each other. The power satisfies the dc voltage requirements of the computer as you learned in chapter 4. Figure 5-1 shows a typical block diagram of a computer. To complete the computer system, the computer uses instructions to perform its operations. Through the man/machine interfaces, you can control the computer's operations to perform maintenance.

In this chapter, we discuss the CPU and buses. In chapter 6, we discuss memory. In chapter 7, we discuss input/output and how the computer interfaces externally with other computers, peripherals, and subsystems. In chapter 8, we examine computer instructions and the man/machine interface.

You can find a computer's functional areas and their operations, functional descriptions, logic implementation, interpretation of logic, and functional schematics in your computer's technical maintenance or owner's manuals. The technical manuals and MRC documentation provide you information on the required and/or recommended tools (standard and specialized), test documentation, and test equipment to perform preventive maintenance. The technical manual or owner's manual documentation provides information to perform all aspects of corrective maintenance. This includes test documentation and procedures; test equipment; and tools for disassembly, assembly, and repair. Repair tools include

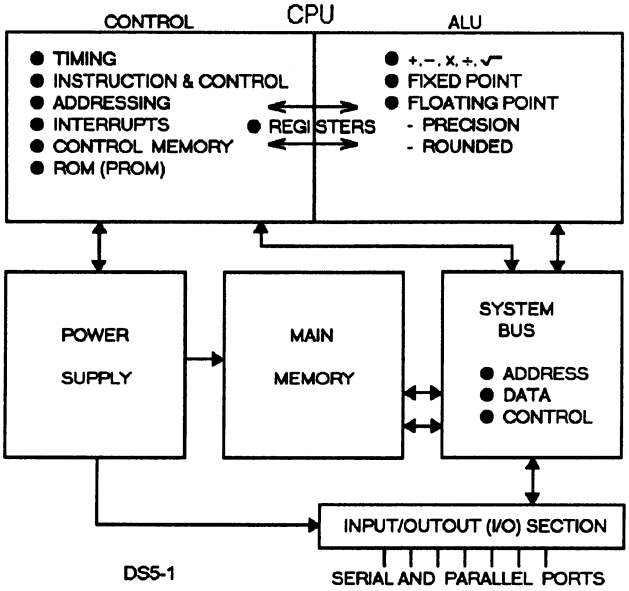

Figure 5-1.—Example of a typical block diagram of a computer.

standard and specialized tools. The specialized tools include solder and solderless repair tools.

Become familiar with your computer's publications and required documentation before you jump into the computer's hardware. This will enhance your abilities as a technician. To perform this job effectively, you must understand how a computer is organized internally. You must be able to recognize the functional areas and what their capabilities are. You must understand how buses function internally to transfer information internally.

The CPU is the computer's brain. All the computational operations (logical and arithmetic) and operational decisions are made in the CPU. The CPU controls all computer operations. The organization of the central processor becomes increasingly more complex as you move from a relatively simple microprocessor to a mainframe computer. But basically CPU functions are the same whether you are talking about a mainframe, a minicomputer, or a microcomputer.

The CPU comprises two interacting sections: the control section and the arithmetic logic unit (ALU). The control section directs the sequence of CPU operations, interprets the instructions, and provides the timing and control signals to carry out the instructions. The arithmetic logic unit implements arithmetic and/or logical operations required by these instructions. The CPU generally consists of timing circuits, registers, translators, selectors, comparators, adders, and subtractors.

After completing this chapter, you should be able to:

- **Recognize the internal parts and functions of a computer**
- **Describe how a control section of a CPU operates**
- **Describe how the functions of the arithmetic logic unit (ALU) are performed**
- **Describe the types of buses and how they operate**

TOPIC 1—CONTROL SECTION

Like a traffic director, the control section decides when to start and stop (control and timing), what to do (program instructions), where to keep information (memory), and whom to communicate with (I/O). It controls the flow of all data entering and leaving the computer, from the beginning to the end of operations. It does this by communicating or interfacing with the ALU, memory, and I/O areas (fig. 5-2). It is also capable of shutting down the computer when the power supply detects abnormal conditions. In some computers it sends a signal to the control section to initiate computer shut-down.

Specifically the control section manages the operations of the CPU, be it a single chip microprocessor or a full-size mainframe. The control section of the CPU provides the computer with the ability to function under program control. Depending on the design of the computer, the CPU can also have the capability to function under manual control through man/machine interfacing. The man/machine interface operating modes, the operations, and the functions, along with the control section, will allow you to control the operations and perform maintenance on the computer(s). NEETS Module 13, *Introduction to Number Systems and Logic Circuits*, and chapter 4 of this volume provide an excellent review of some of the circuits used in the control section.

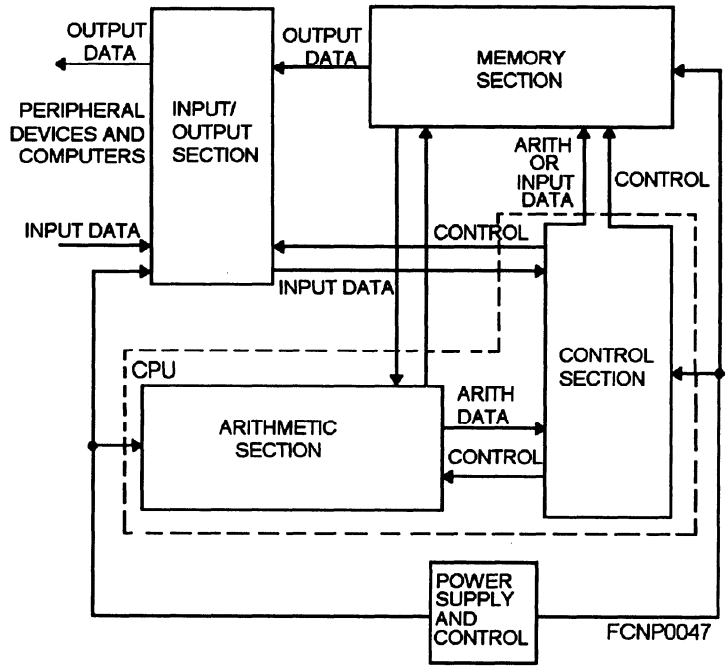

Figure 5-2.—Representative block diagram showing the relationship of the control section to the other functional areas of a computer.

The control section consists of several basic logically defined areas. These logically defined areas work closely with each other. They are the basis for the operations of the control section in most computers. They include:

- Timing
- Instruction and control
- Addressing
- Interrupts
- Control memory
- Cache memory
- Read-only memory (ROM)

TIMING

Timing in a computer regulates the flow of signals that control the operation of the computer. Without timing, events in a computer would not take place. The computer's operations rely on both **synchronous** and **asynchronous** operations. Synchronous operations means that certain events happen at regularly timed intervals. An example of this is the computer's master clock. Asynchronous means that the completion of one event triggers the next event. An example of this is the execution of instructions located sequentially in memory. After an instruction is executed, the next instruction cannot be executed until the program counter has been incremented to fetch it. Timing gets the computer going. Timing circuits are used throughout the computer, as you will see when we discuss each of the functional areas.

Not all computers rely on a sophisticated timing system. Some timing systems are very simplistic and rely only on the computer's master clock and one or two other timing signals derived from the master clock to start and stop events. Still other more sophisticated computers rely on the master clock and timing circuits in each of the functional areas to start and stop operations.

Some of the more common timing circuits you will encounter include the following:

- Master clock
- Main timing chain
- Main timing signals
- Timing sequences
- Sequence enables and control
- Real-time clock
- Monitor clock
- Programmable interval timers
- Arithmetic timing

Figure 5-3 is an example block diagram of timing circuitry used in a computer's CPU.

Master Clock

From our discussion in chapter 4, you learned that the master clock can be either a single- or multiple phase master clock. A single-phase master clock can then be used to trigger a single-shot multivibrator that is used throughout the computer to enable and disable circuits in whatever sequence is necessary to properly execute the computer's operations. Multiple-phase master clocks can use a pulse generator or delay line oscillator to generate two or more clock phases. A delay line oscillator will generate two basic clock phases and any additional phases are derived from taps on the delay line oscillator.

Whether a pulse generator or delay line oscillator is used, they generate multiple phases sometimes referred to as odd θ1 (CP1) and even phases θ2 (CP0) or lettered phases (θA, θB, or (θBA).These phases from the master clock are then used to initiate the main timing chain flip-flops. The master clock in a computer can be suspended under certain conditions; the way it can happen varies with the type of computer. With a microcomputer, it is usually done by removing power to the computer. With a larger mainframe or minicomputer, you will need to remove the power works, too. However, certain types of **HOLDS, MASTER CLEARS,** and operating **MODES** selected at a console can also suspend master clock oscillations. Refer to your computer's technical manual for details. Refer again to figure 5-3 for an example.

Main Timing Chain

The main timing chain consists of flip-flops arranged in a ring counter. It is used to count master clock phases. The flip-flops used in the main timing can be set and cleared by the two basic master clock phases and any additional master clock phases. The design of the computer determines how this is accomplished. The main timing (MT) chain is often

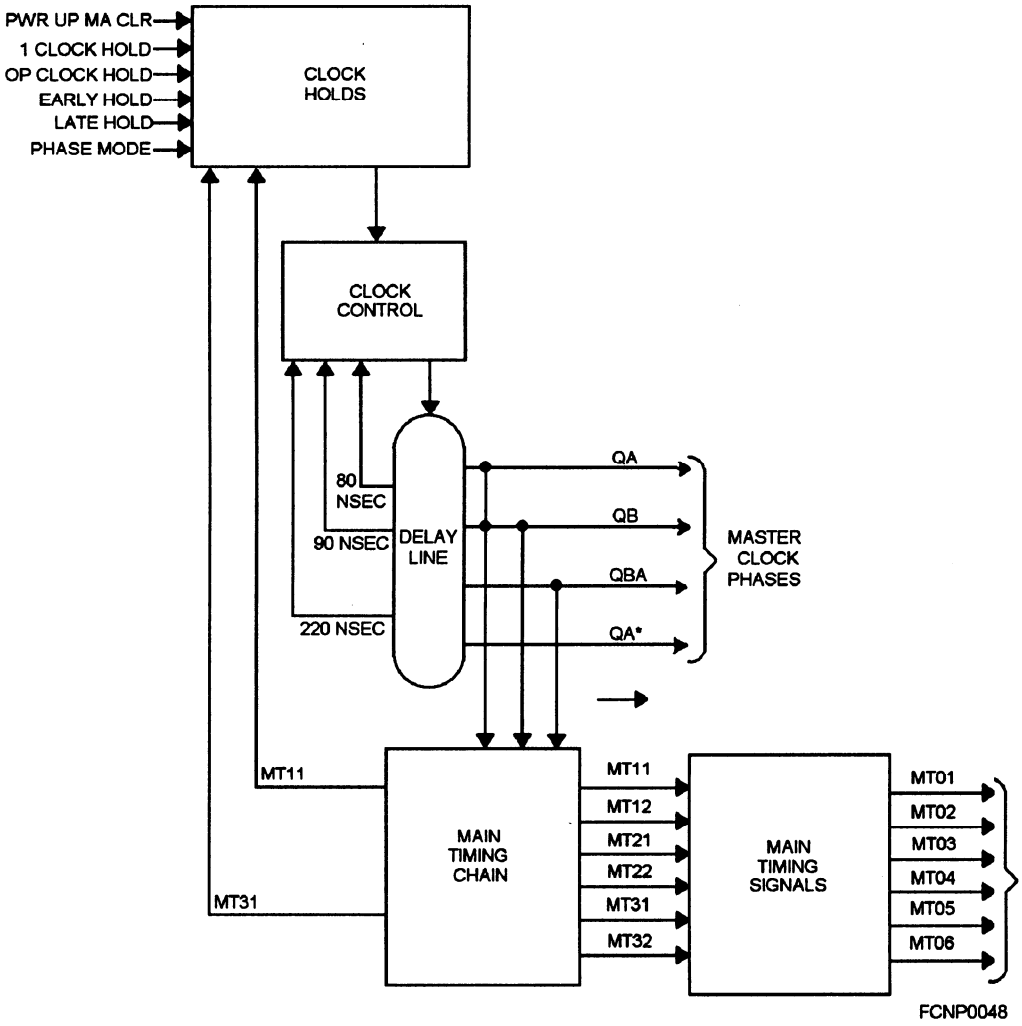

Figure 5-3.—Example block diagram of timing circuitry used in a computer's CPU.

designed so that the odd flip-flops (MT11, MT21, and so on) are set and cleared by odd phases or lettered phases and the even flip-flops (MT12, MT22, and so on) are set and cleared by the even or lettered phases. The timing chain uses the set and/or clear sides of the flip-flops to enable and disable circuits throughout the computer and to generate main timing signals (phases) such as MT0l or MT02. Main timing signals can be used to generate other commands, such as starting arithmetic timing for computers with more sophisticated mathematical operations.

Main Timing Signals

Main timing signals are used in the CPU to enable and disable circuits or generate command enables that are used for control or arithmetic operations. The majority of data transfers affecting the registers and associated circuitry in the control section derive their enables from main timing signals. An example is a main timing signal used to generate a command enable such as sending data from one register to another.

Timing Sequences

Timing sequences are used to issue a series of commands to perform a particular instruction or operation. The minimum number of sequences per instruction or operation is determined by the requirements of the computer. An example is the command to enable an instruction sequence, which is used to acquire the instruction for translation.

Some computers have separate control sections for each functional area. In that case, each function will operate independently of the others. That is, a computer that uses a controller for I/O operations has its own master clock/main timing chain/main timing signals, which are independent of the CPU's master clock/main timing chain/main timing signals.

Sequence Enables and Control

Circuitry to control the sequence enables and to generate commands depends upon the type of instruction and method of addressing.

Real-Time Clock (RTC)

The real-time clock (RTC) is used to keep track of units of real time. The RTC can be loaded, read, enabled, and disabled by machine instruction. The register itself is incremented at a rate determined by the RTC oscillator circuit setting or the external RTC input frequency.

The RTC is only incremented when the CPU is running. It allows the computer, through machine instructions, to keep track of the passage of time using readily processed units of time. To prevent register overflow from causing errors in the timekeeping process, most RTCs generate register-overflow interrupts when the register contents increment around to zero (change from all ONEs to all ZEROs). The RTC can be enabled and disabled, and updated internally or externally.

Monitor Clock

The monitor clock register is used to keep track of time intervals by counting down from its loaded value to zero. The monitor clock can be loaded, enabled, or disabled by machine instruction. The monitor clock is decremented in the same manner as the RTC is incremented and only when the computer is running. When the enabled monitor clock reaches zero, a monitor clock interrupt is generated. A monitor clock interrupt usually indicates that a designated computer operation timed out before it was properly completed. This usually occurs when memory or I/O cannot honor a request for reasons of priority or hardware failure. There must be a time limit established to release the hold on CPU main timing or an indefinite period of inaction could occur. By using the monitor clock register to keep track, a time limit is imposed.

Programmable Interval Timers

For those microprocessors that do not have an RTC or monitor clock registers, there is an additional logic chip available called a programmable interval timer. This chip provides up to three counters or count registers that are software controlled. These registers can perform the RTC, the monitor clock, or any other time interval measurements.

The timer communicates with the CPU over the control and data buses. The count registers are independent of each other, addressable (0, 1, or 2), and can be loaded with count values or have their current values read and sent to the CPU. These counters are decrementing or down counters only. They operate off of separate clock signal inputs so they can be configured to count at the same or different clock rates. They can also be programmed to interrupt the CPU when the count in a selected register reaches zero.

Arithmetic Timing

Arithmetic timing is initiated by a command from the CPU's main timing chain. How far arithmetic timing advances is dependent upon the specific instruction.

INSTRUCTION AND CONTROL

The instruction execution and control portion of the control section includes the combinational and sequential circuits that make up the decision-making and memory-type functions. First we discuss some of the functions, operations, operand addressing, and operating levels. We include those items most common in all computers and any that are unique to a specific type of computer.

Instruction and Control Functions

In chapter 4 we discussed the circuits that are used by computers. In this topic we discuss some of the more common functions used by these decision-making and memory-type circuits to execute instruction and control operations. Some of the more common functions of the circuits in this area include the accumulators, index registers, instruction register, program counter, and status indicating registers.

The registers (memory-type functions) work with decision-making functions (primarily data routing circuits) to channel the data inside the computer. Their functions are many in the CPU; therefore, we do not go into detail. Refer back to chapter 4 for their basic functions. These data routing circuits are capable of providing input to the registers and/or using their outputs to route data elsewhere in the computer. Among some of the data routing circuits included in the CPU's control section are the following:

- Adders
- Command signals (enables)
- Comparators
- Demultiplexers
- Selectors
- Translators

These are by no means all the functions contained in all computers, but they represent a general overview of the common functions needed to execute instructions and control operations.

Let's look at the more common functions of the memory-type circuits that the CPU uses.

ACCUMULATORS.— Located in the CPU are a number of general-purpose **registers** called accumulators that are used to temporarily store data or memory addresses. They are generally the same length (number of bits) as a memory word. There are typically 8-, 16-, or 32-bit accumulators, numbered from 0, depending on the size and type of computer or microprocessor.

These registers are accessible to a computer programmer. In other words a programmer can control, by machine instruction(s), what data is placed in these registers and what manipulations take place on the data. In addition to the operation (op) code, instructions contain one or two multibit fields that specifically identify the accumulator register to be operated upon.

In older computers each bit position's flip-flop circuit had indicator lamps to indicate the contents of the register to the computer programmer/technician. In the newer computers, the majority of registers are noting more than memory addresses in local storage areas. The register contents, however, are still accessible to the technician through the computer's man/machine interface.

INDEX REGISTERS.— Most CPUs contain a number of index registers (8-, 16-, or 32-bit). Index registers are addressable registers that are used for two purposes: **address modification** and **counting.** The value contained in a particular index register can be used to modify the operand address of a machine instruction without changing the instruction itself in memory. In this way a single instruction can be used to specify a large number of operands, indirectly.

The count in an index register can also be modified by fixed values (incremented or decremented) to control program repetitions or iterations.

INSTRUCTION REGISTER.— To translate and execute the instructions, the outputs of the instruction register are fed to logic circuits **(selectors** and/or **translators)** that are used to translate the binary codes into **commands** for the CPU to execute (fig. 5-4).

PROGRAM COUNTER.— The program counter controls the selection of machine instructions. It holds the address of the next instruction to be executed. **Adders** and **registers** are used to perform this function.

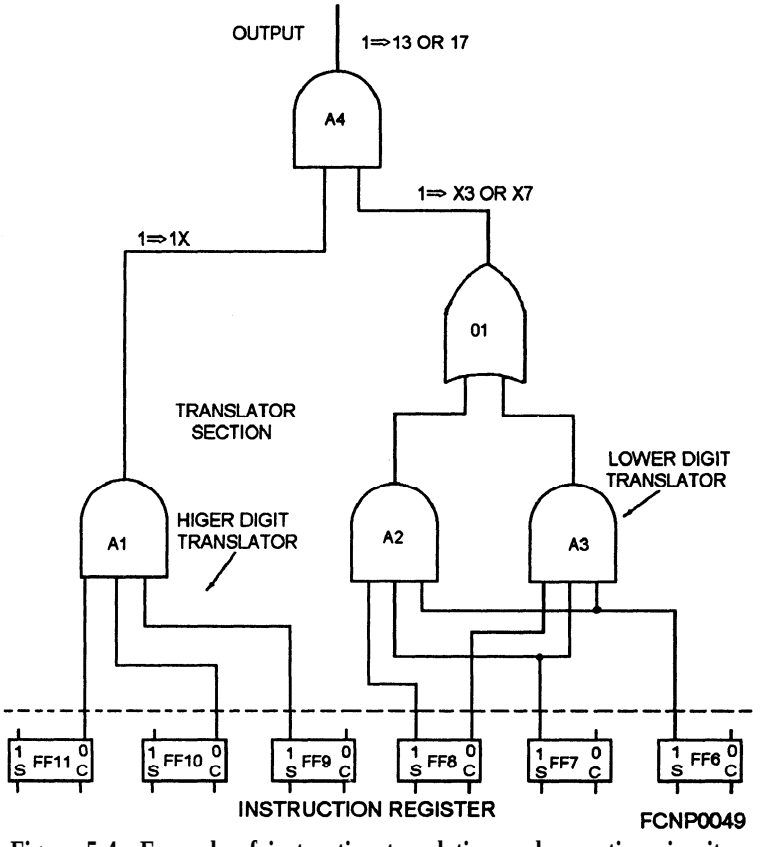

Figure 5-4—Example of instruction translation and execution circuitry.

STATUS INDICATING REGISTERS.— The CPU must have some way to monitor the status of the computer's internal operations. The name of these register or registers may differ between computers, but the general functions performed are the same. Some of the most common names are as follows:

- Condition code
- Status and control
- Program status
- Active status
- Flag

These registers use the condition of individual bits in the register to indicate the status of operations in the computer (fig. 5-5). Within the register, individual and sometimes groups of bits (2 or 3 bits) are hardwired to the computer logic. The 1 or 0 value in each bit position indicates the status of a particular activity or special function of the computer.

The specific activities monitored by these registers varies between computers, but consist of the following general areas: **arithmetic operation or comparison results** (carry, overflow, zero, negative, and so forth)

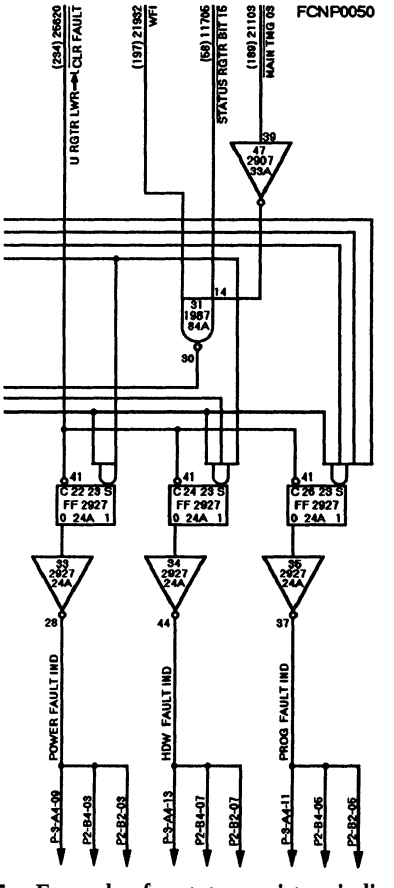

Figure 5-5.—Example of a status register; indication of a program fault.

(fig. 5-6), a variety of **interrupt conditions, task or executive state status,** and **hardware status** (memory lockout, hardware faults, and so forth). These registers are often used with instructions where branching conditions are used to change the sequence of instruction execution.

The status indicating registers' contents can be sensed, loaded with new data bits, or stored into memory by machine instruction. Many machine instructions, particularly branching instructions, are designed to sense the condition of specified register bits to determine how the instruction itself is to be executed. Other instructions are designed to modify the contents of the register(s) to change state (executive or task) or to enable/disable classes of interrupts; this is accomplished by indexing. The contents of the status indicating register(s) is/are normally stored into memory as part of the interrupt processing operation.

Instruction and Control Operations

The control portion of the CPU for computers is responsible for fetching, translating, and executing all instructions (fig. 5-7). The CPU calls up or reads the instructions one at a time either from consecutive addresses or as dictated by the program from main memory or read-only memory (ROM). The general process of execution of a machine instruction can be divided into four major parts: fetch (read) the instruction, update the program counter or equivalent, translate the instruction, and execute the instruction specified by the function or op code.

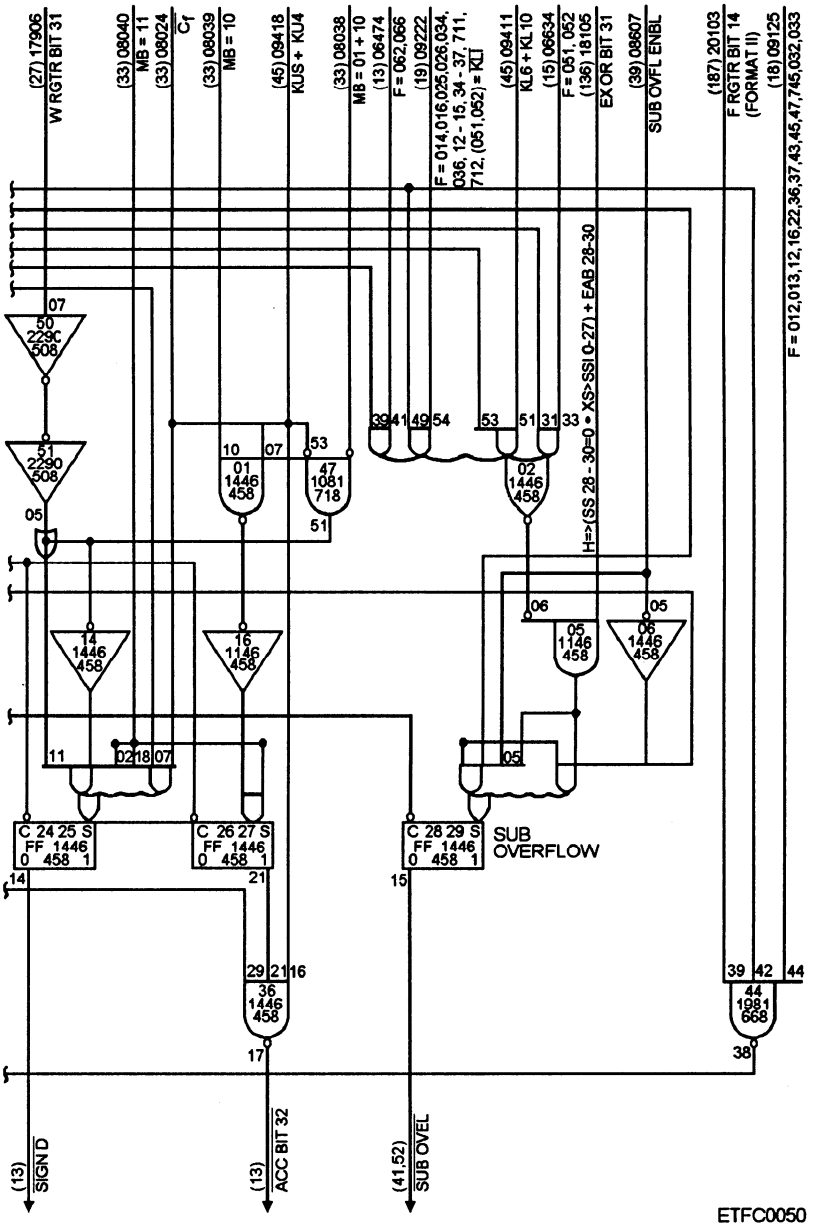

Figure 5-6.—Example of an arithmetic detecting circuit used to indicate a subtraction overflow condition.

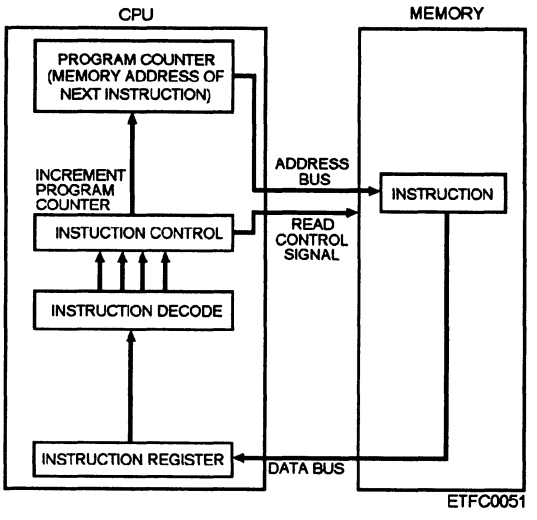

Figure 5-7.—Basic operation of a fetch and decode (translate) of an instruction in a CPU.

FETCH (READ) THE INSTRUCTION.— The instruction is fetched by reading the instruction from the memory (main memory or ROM) address specified by the contents of the program counter or equivalent. The instruction is temporarily stored in an **instruction** register, while the program counter is being incremented to the next instruction's address.

UPDATE THE PROGRAM COUNTER.— The program counter controls the selection of the instruction. The program counter contains the memory address of the next machine instruction to be executed. Most of the time machine instructions are executed sequentially. The program counter is incremented to the address of the next instruction. Usually an **index adder** is used to perform this function. When an instruction is completed, the new count in the program counter points to the next instruction to be fetched from memory and executed in turn.

The memory word size of the computer has an effect on the value that is used to increment the program counter. For those computers in which the majority of instructions are contained in one memory word, the program counter is incremented by one (1) for each instruction. For computers with smaller memory words (8-bits), instructions are often assembled from several sequential bytes and the program counter must be incremented by a value that will point to the first byte of the next instruction to ensure correct translation of that instruction's operation code.

There are times, however, when a change in the sequence of instruction execution, called **branching** or jumping, is required. Branching or jumping can be accomplished through the man/machine interface by using switches on the controlling consoles. Examples

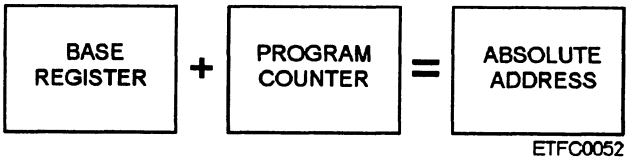

Figure 5-8.—Block diagram of an operation to determine an absolute address.

are the stop and jump switches. In these cases instead of being incremented, the address in the program counter is changed to anew address to start sequential execution of a different section of machine instructions in the program. Branching or jumping can also be accomplished through program instructions.

In some computers, the program counter contains the **relative** or **offset address** of the instruction being executed. An additional set of registers called **base registers** are used to provide the base address of a block of memory. The program counter value must be added to a selected base register value (fig. 5-8) to determine the **absolute address** of the next sequential instruction.

TRANSLATE THE INSTRUCTION.— An instruction register holds the machine instruction while it is translated by other CPU logic (translators). The binary data that makes up the instruction op code determines the operation the CPU is to perform. The derived function codes are then sent to other parts of the control section of the CPU to execute the instruction. The translation of the instruction determines which command sequences will be used to execute the instruction.

EXECUTE THE INSTRUCTION.— Execution of the instruction will generate **command enables** that are used throughout the computer to transfer data between registers and other parts of the computer. The logic consists of gating and amplifying circuits, which produce or inhibit **control signals** appropriate to the combination of conditions at their inputs. The controlling conditions are supplied by the timing circuits (master clock, main timing chains, and timing sequences) and function code translator and associated circuitry (selectors, registers, adders, and comparators).

An execution technique used in newer microprocessors contains a logic assembly called an **instruction queue.** It is used to speed up computer operations and increase efficiency. The instruction queue allows the microprocessor to fetch a number of sequential instructions or instruction bytes and hold them in a queue for execution by the execution unit of the microprocessor. The instructions are fetched by the bus when the memory section is available for access and in some cases pretranslated while the processor is

executing other instructions. Instructions or instruction bytes are added to the rear of the queue until the queue is full. When the execution unit has completed an instruction, it simply takes the next instruction or several instruction bytes from the front of the queue.

Instruction Operation Levels

The CPU executes instructions at two levels or states: the executive state and the task state. Data bits in the status indicating registers(s) are used to select the desired **active state.**

- Executive state —Executive state, also called interrupt state, instructions are designed to process what are known as **executive functions** (primarily I/O and interrupt processing) for multiprogramming operations. These functions are included in the operating system programs. There may be as many as four separate executive states in newer computers, one for each class of interrupts.

- Task state —Task state instructions execute what are called **application functions.** These functions actually perform the work, such as solving the fire control problem in a CDS/NTDS platform or computing a sonobuoy pattern on a TSC platform.

The majority of machine instructions can be executed in either the task or executive states. There are a limited number of instructions that can be executed only in the executive states. An example is **privileged instructions** that are part of **interrupts,** which you will learn more about later in this topic.

Those computers that have task and executive states have at least one set of addressable registers for each state. These addressable register types (accumulators, index registers, base registers, and the like) are only accessible by machine instruction when the computer is in the applicable state. The register sets are enabled and disabled automatically as the computer changes states. In computers with four executive states, there are five sets of addressable registers, one for the task state and one for each executive state.

INSTRUCTION OPERAND ADDRESSING

Addressing is the process of locating the operand (specific information) for a given operation. It is similar to the process of obtaining your address so that information can be sent to you. Once the computer knows where to obtain the location of the operand, the instruction can be carried out. If for instance, the operand is in memory, the addressing technique determines how to obtain the memory address of the operand and how to use this address to locate the operand and fetch it. If the operand is in one of the CPU's registers, addressing is the means by which the instruction specifies the selected register and the operand is fetched. Because the length of instructions and the number of bits per memory cell vary between types of instructions and computers, there is a variety of ways the operand may be obtained.

INTERRUPTS

Up to this point we have covered timing and instruction control and execution. The following information is designed to tie together the overall operation of the computer through the study of interrupts and interrupt processing. We first cover the definition of an interrupt and the types and classifications of interrupts you will encounter in computer systems. Then, we cover how computers handle interrupts and what happens within the computer hardware and software.

An interrupt is defined as a break in the normal flow of operation of a computer caused by an **interrupt signal.** The break occurs in such a way that the operation can be resumed from the point of the break at a later time with exactly the same conditions prevailing.

Interrupts are a method of diverting the attention of the computer from whatever process or program it is performing to the special condition or event that caused the interrupt signal. Interrupts allow the computer to respond to high priority demands and still be able to perform normal or lower priority processing. When the condition that caused the interrupt signal to occur has been addressed or processed, the computer's attention can be returned to the process or program it was executing before the interrupt with the **exact same conditions prevailing.** Interrupts can occur either **asynchronously** or **synchronously** within the CPU program. The handling of a synchronous interrupt occurs with the actual event that caused the interrupt; whereas the handling of an asynchronous interrupt may occur much later in time than the actual event that caused the interrupt. We discuss the classification, types (micro, mini, and mainframe computers), priorities, codes, and handling processes of interrupts.

Classifications of Interrupts

There are two major classifications of interrupts: internal interrupts and external interrupts.

- Internal interrupts —Internal interrupts occur as a result of actions or conditions within the sections of the computer (CPU, IOCs, or memory). Internal interrupts tend to indicate the completion or termination of I/O operations, or the ending of defined time periods; or they signal some type of error.

● External interrupts —External interrupts are received from external peripheral devices. They are used to synchronize the execution of computer programs to the readiness of the peripheral device to transmit or receive data. They are also used to identify peripheral equipment problems/errors to the computer.

Now let's look at how interrupts work in each major type of computer.

MICROCOMPUTER INTERRUPT TYPES.— The microcomputer receives both internal and external interrupts. Internal interrupts are received from the real-time clock, system clock, and other conditions that effect the operation of the microprocessor. External interrupts are received from disk drives, CD-ROM drives, sound boards, etc. These are classified as external interrupts, even though the devices are physically installed in the microcomputer case. Microcomputer interrupts fall into two basic categories: maskable and non-maskable. The CPU of the microcomputer has two interrupt signal lines, one for each category of interrupt.

External hardware interrupts are maskable interrupts. The interrupt request signal indicates the presence of one or more of these interrupts. The specific interrupt type is defined by accompanying interrupt code words. The interrupt code and a ROM or programmable ROM (PROM) lookup table are used to direct the processor to the address of the interrupt processor program for the particular interrupt type. Maskable interrupts can be masked out or locked out for short periods of time by the software to allow the CPU to perform critical operations. The programmer is responsible for ensuring that interrupts are managed in a timely manner.

Nonmaskable interrupts cannot be masked out. They are used for conditions that require immediate attention by the microcomputer. Examples include interrupts from the internal hard disks, modems, fax cards, and sometimes a power out-of-tolerance condition. If this feature is available, a power out-of-tolerance condition will force the microcomputer to execute its save data program.

The **interrupt request (IRQ)** line provides the input signal path for all interrupts. If the interrupt enable bit in the status indicating register is set, the interrupt is processed at the end of the current instruction cycle. If the interrupt enable bit is clear, the interrupt signal is ignored by the microcomputer and the next sequential instruction is executed.

Each hardware interrupt has a unique IRQ channel assigned. Some of these channels are preassigned and cannot be changed, while several are available for the user to install additional hardware into the microcomputer. Table 5-1 lists the hardware interrupt channels used by most microcomputers. Note that in

Table 5-1.—Common IRQ Assignments for Microcomputers

IRQ Channel	INTERRUPT FUNCTION	BUS SLOT/SIZE
0	System Timer	No
1	Keyboard Controller	No
2	Cascade to IRQ 9	Yes/8 or 16 bit
3	COM2 or COM4	Yes/8 or 16 bit
4	COM1 or COM3	Yes/8 or 16 bit
5	Parallel Port 2 (LPT2)	Yes/8 or 16 bit
6	Floppy Disk Controller	Yes/8 or 16 bit
7	Parallel Port 1 (LPT1)	Yes/8 or 16 bit
8	Real-Time Clock	No
9	Available 9May appear as IRQ 2	Yes/8 or 16 bit
10	Available	Yes/16 bit
11	Available	Yes/16 bit
12	Motherboard Mouse Port	Yes/16 bit
13	Math Coprocessor	No
14	Hard Disk/Primary IDE Controller	Yes/16 bit
15	Secondary IDE controller/Available	Yes/16 bit

Table 5-1, IRQ5 is assigned to parallel port 2; this port is generally available in most microcomputers and is commonly used by most sound cards. When microprocessors expanded from 8-bit to 16-bit processors, the amount of hardware supported also grew. This required the addition of more IRQ channels. Manufacturers added an additional 8-channel processor and cascaded them by connecting IRQ2 on processor to IRQ9.

The latest development in microcomputer technology concerning interrupt processing is the Plug-n-Play feature. A true plug and play system requires three components to work together; the hardware, the BIOS, and the operating system. During the power-on cycle of computers that are Plug-n-Play capable, the firmware contained in the basic input/output system (BIOS) interrogates each component in the system to determine the type of board, IRQ channel requirements, DMA channel requirements, and ROM requirements. The board responds with the specifications it requires, then the BIOS assigns IRQs, DMA, ROM resources, etc., to all the boards, ensuring that there are no conflicts. The functions of the BIOS are covered in detail later in this chapter. This process is repeated every time the computer is turned on. Controllers that are not Plug-n-Play compatible can be installed by using the standard configuration program and locking the resource to those unique settings.

MINI AND MAINFRAME INTERRUPT TYPES.— Within larger computers, interrupts are divided into a number of separate classes. Multiple classes of interrupts are needed because there are several levels of processing within these computers and many different types of operations and conditions that have to be monitored. Some operations and conditions are more important than others.

There are generally three or four classes of interrupts, which we designate class I, II, III, and IV. Interrupts are prioritized by these classes and by the types of interrupts within a class. Class I interrupts are the highest priority or most important interrupt class as far as the computer is concerned. The other classes (II, III, and IV) are in turn lower in priority than Class I.

Class I Interrupts.— Class I interrupts function during all computer operations; in other words, they will interrupt any computer program or instruction. These are the highest priority interrupts. Known as fault and hardware or hardware error interrupts, these interrupts indicate there is a serious hardware problem with the computer, or more accurately within the CPU or its communication buses. The following are some of the more common class I interrupts:

- Power fault or power tolerance
- Memory parity errors
- Memory resume errors
- Bus communication errors

The most common class I interrupt is the **power fault** or **power tolerance** interrupt. This interrupt indicates that the power supply voltage has fallen below a certain tolerance level and that the computer should execute its power failure processing routines before there is a total loss of power. The actual routines will vary from computer to computer based on the device's automatic restart and backup storage power capabilities.

Class II Interrupts.— Class II interrupts are used to identify faults and errors within the CPU or IOC instruction execution and program timing processes. These **software interrupts** can indicate the following conditions:

- Execution of illegal instruction operation (op) codes (CPU or IOC instructions)
- Execution of privileged instructions in the task mode
- Floating-point math underflow or overflow conditions
- Real-time clock (RTC) overflow
- Monitor clock timeouts

Class III Interrupts.— Class III interrupts are primarily **I/O operation interrupts.** They indicate such functions as the following:

- External interrupts
- Input or output chain interrupts
- Intercomputer timeouts
- Input data ready or output data ready interrupts

Class IV Interrupts.— In some computers, there is a class IV interrupt that indicates **executive state entrance.** In others, the executive state entrance is a class II interrupt. A limited number of instructions can be executed only in the executive states. Among them are privileged instructions.

MINI AND MAINFRAME INTERRUPT LOCKOUT OF CLASS I, II, III, AND IV TYPES.— Computers that operate with different levels of interrupts are equipped with the logic circuitry to

lockout or disarm classes of interrupts and often specific interrupts within a class. Lower levels of interrupts (class II through IV) can be **locked out** (disarmed) or **enabled** (armed) by machine instruction. The terms *prevent/allow* are also used in place of enable/disable with some computers. The lower priority interrupts are locked out so that they do not interfere with higher level computer operations (executive state or class I interrupt processing) while they are in progress.

There are usually several specific class I interrupts that cannot be locked out by instruction. These interrupts would normally include any of the following:

- Power fault
- CPU instruction fault
- IOC instruction fault interrupts

INTERRUPTS AND INTERRUPT CODES.— Interrupt signals, as a rule, cause the computer to reference a freed address in memory and execute the subroutine (a series of instructions) identified by the contents of the address. The interrupt signal only identifies the class of interrupt. Multiple interrupt types within a class are usually defined by an accompanying **interrupt code** or **interrupt code word.**

In older and smaller computers, the interrupt code parallels the interrupt signal. In other words both the interrupt signal (class I, II, or 111) and identifying code are received and processed by the CPU at the same time. Since the interrupt processor tends to lockout interrupts of the same class, this process tends to hold up or even lose interrupts of the same or lower priority classes that occur while the first interrupt is being processed.

Newer computers retain multiple interrupt codes of the same class in an **interrupt stack** or **interrupt queue,** usually contained in the I/O section. There usually is a stack or queue for each interrupt class (I, II, or III). Interrupt queues store their codes in first-in, first-out (FIFO) order.

The interrupt signal would indicate to the CPU the presence of at least one interrupt of the particular class. The stack and queue arrangements allow the CPU to sample the interrupt codes at its convenience. As each code is processed, it is removed from the stack or queue until the stack or queue is empty. The interrupt signal would only drop if the stack or queue becomes empty. New interrupt codes would simply be added to the stack or queue as they occur. An empty stack or queue would generate an interrupt signal when the first new code is added to the stack or queue by the I/O circuits.

INTERRUPT HANDLING PROCESS.— CPUs follow a specific sequence of events when processing an interrupt. Remember interrupt processing has priority over normal program execution. We discuss the general interrupt handling process in order of its sequence. Figure 5-9 illustrates the general sequence

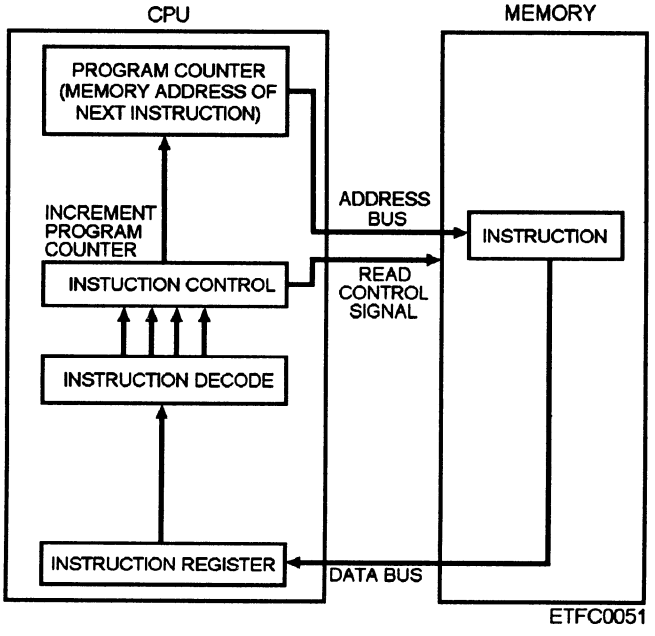

Figure 5-9.—General sequence of an interrupt response.

of an interrupt response by the CPU. Refer to this figure as we describe the process.

Terminate Current Program Execution.— Computers are not designed to instantly stop all current operations when an interrupt signal is received. They do not halt the current operation until the machine instruction (macro or micro) being processed has been completed. Interrupt terminations effectively occur between instructions. There is usually a check for interrupt signals at the end of the current instruction execution cycle. In our example, an interrupt is received during the execution of the third instruction.

At this time, the program counter has been incremented to the next instruction's address, and all register operations are complete from the execution of the instruction in the instruction register, the third instruction. The program counter reflects the address of the next instruction in the current program and the register contents are stable. It is at this point that the interrupt process will be initiated.

Lock Out All Interrupts.— The first event that takes place in interrupt processing is the locking out of all new interrupts. This is done to protect the integrity of the process that ensures returning to the same conditions after processing the interrupt. There are a few machine instructions and other processes that must be performed to save the current register data so that it can be restored to the preinterrupt conditions. The interrupt lockout prevents any new interrupts from interrupting this process and potentially losing data or even worse losing track of where the computer was in the interrupted program.

Store Program and Register Data.— Once all interrupts have been locked out, the computer can store the current process's register data in the applicable memory locations. Each class of interrupt is assigned a block of memory locations to store at least the following register contents: program counter and status register(s).

The program counter data will allow the interrupted process to be restarted as if the next instruction is being executed as in normal operation. The status register contents are saved to be able to reinstate the computer's operational status at the time of the interrupt once the interrupt has been processed. In our example, the data from the three previously executed instructions is stored in memory. The address of the fourth instruction of the current program is also saved.

In newer computers, the accumulator, index, and other addressable registers do not require saving since there is a separate register set for each task and executive state. When a new state is entered, the instructions being executed can only address or modify the registers assigned to that state. Any other task or executive state registers are disabled and their contents are protected until the appropriate state is reentered.

Retrieve Interrupt Processor Data.— After the register data is saved, the new executive state's registers are loaded with the interrupt processor program data. The program counter is loaded with the starting address of the processor program (instruction number 1 of the interrupt routine), the status register(s) is/are loaded with the operational status data required by the program. The interrupt processor data for each class of interrupts is stored in an assigned block of memory cells where it can be retrieved for each interrupt.

Enter Executive State and Enable Desired Interrupts. — The loading of the status register(s) allows the computer to enter the required executive state and enable the interrupts that can in turn interrupt the interrupt processor. The data bits loaded into the status register(s) effectively change the executive state class (I, II, III, or IV), and enable the active status register set.

The new status register bits also set or clear interrupt lockouts to enable or disable specific interrupt classes. The new data in the status register(s) would only enable higher priority interrupts than the interrupt being processed.

Execute Interrupt Processor Program.— The address in the active state's program counter will now allow for the execution of the interrupt processor program, instruction number 1 of the interrupt routine. The interrupt processor samples the interrupt code words and determines the appropriate action in response to the interrupt.

Return to Original Process.— Upon completion of the interrupt processor routine, the active state will be switched to the next lower state, either task state or a lower priority executive state, and the program counter and status register(s) for that state will be reloaded with the saved data. The program counter can then call up the next sequential instruction (instruction number 4 of the current program) in the interrupted process and the program will continue as if no interrupt had occurred. The computer will normally return to the task state program only when all executive state procedures have been completed.

CONTROL MEMORY

Control memory is a random access memory (RAM) consisting of addressable storage registers. It is primarily used in mini and mainframe computers as a temporary storage for data. Access to control memory data requires less time than to main memory; this speeds up CPU operation by reducing the number of memory references for data storage and retrieval. Access is performed as part of a control section sequence while the master clock oscillator is running.

The control memory addresses are divided into two groups: a task mode and an executive (interrupt) mode. Addressing words stored in control memory is via the address select logic for each of the register groups. There can be up to five register groups in control memory. These groups select a register for fetching data for programmed CPU operation or for maintenance console or equivalent display or storage of data via a maintenance console or equivalent. During programmed CPU operations, these registers are accessed directly by the CPU logic. Data routing circuits are used by control memory to interconnect the registers used in control memory.

Some of the registers contained in a control memory that operate in the task and executive modes include the following:

- Accumulators
- Indexes
- Monitor clock status indicating registers
- Interrupt data registers

CACHE MEMORY

Cache memory is a small, high-speed RAM buffer located between the CPU and main memory. Cache memory buffers or holds a copy of the instructions (instruction cache) or data (operand or data cache) currently being used by the CPU. The instructions and data are copies of those in main memory.

Cache memory provides two benefits. One, the average access time for CPU's memory requests is reduced, increasing the CPU's speed by providing rapid access to currently used instructions and data. Two, the CPU's use of the available memory bandwidth is reduced. This allows other devices on the system bus to use the memory without interfering with the CPU. Therefore, cache memory is used to speed up the flow of instructions and data into the CPU from main memory.

This cache function is important because the main memory cycle time is typically slower than the CPU clocking rates. To accomplish this rapid data transfer, cache memories are usually built from the faster **bipolar** RAM devices rather than the slower **metal-oxide-semiconductor (MOS)** RAM devices. The RAMs used for cache memory may be either dynamic RAMs (DRAMs) or static RAMs (SRAMs). Cache memories are not part of the memory section and they are transparent to programmers (i.e., not accessible by machine instruction). Their size varies with the type of computer; usually they are no more than 64K.

PROPERTIES OF CACHE MEMORY.— All caches share the following properties:

- A buffered memory or cache memory consists of a small high-speed memory with main memory information. This information may be addresses, data, or instructions. The speed of the small memory is usually on the order of one magnitude faster than main memory, and its capacity is typically one or two orders of magnitude less than main memory.

- A cache memory system requires an **identifier** or **tag store** to indicate which entries of main memory have been copied into it. Such an area is usually referred to as the directory or tag store.

- A cache memory requires a logical network and method of replacing old entries.

- A cache memory uses timing and control.

CACHE PROCESS.— The cache process takes place when a CPU with a cache initiates a memory reference. The address of the needed item is generated and the cache is searched. The method of search depends on the type of cache **mapping** used by the computer system. We can generalize the cache process into three areas as follows:

- Searches —Reads from the cache directory with a hit indicating that the data from the requested address is present, while a miss indicates that the data is not present.

- Updates —Writes to the cache data as well as to the directories with new informationo

- Invalidates —Writes only to the directories; this effectively removes an address that previously resided in cache.

If the particular address is found in the cache, the block of data is sent to the CPU, and the CPU goes about its operation until it requires something else from memory. When the CPU finds what it needs in the cache, a hit has occurred. When the address requested by the CPU is not in the cache, a miss has occurred and the required address along with its block of data is brought into the cache according to how it is mapped.

Cache processing in some computers is divided into two sections: **main** cache and **eavesdrop** cache. Main cache is initiated by the CPU within. Eavesdrop is done when a write to memory is performed by another requestor (other CPU or IOC). Eavesdrop searches have no impact on CPU performances.

CACHE MAPPING TECHNIQUES.— Cache mapping is the method by which the contents of main memory are brought into the cache and referenced by the CPU. The mapping method used directly affects the performance of the entire computer system.

• Direct mapping —Main memory locations can only be copied into one location in the cache. This is accomplished by dividing main memory into pages that correspond in size with the cache (fig. 5- 10).

• Fully associative mapping —Fully associative cache mapping is the most complex, but it is most flexible with regards to where data can reside. A newly read block of main memory can be placed anywhere in a fully associative cache. If the cache is full, a replacement algorithm is used to determine which block in the cache gets replaced by the new data (fig. 5-11).

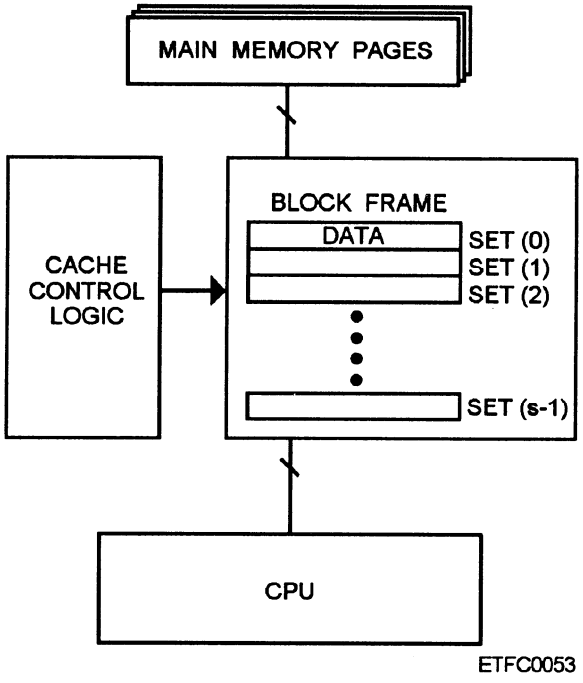

Figure 5-10.—Example of direct mapping used in cache memory.

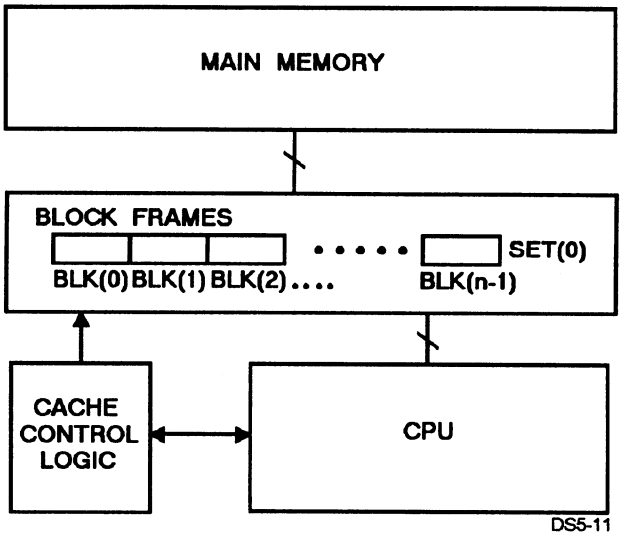

Figure 5-11.—Example of fully associated mapping used in cache memory.

• Set associative mapping —Set associative cache mapping combines the best of direct and associative cache mapping techniques. As with a direct mapped cache, blocks of main memory data will still map into as specific set, but they can now be in any N-cache block frames within each set (fig. 5-12).

CACHE READ.— The two primary methods used to read data from cache and main memory are as follows:

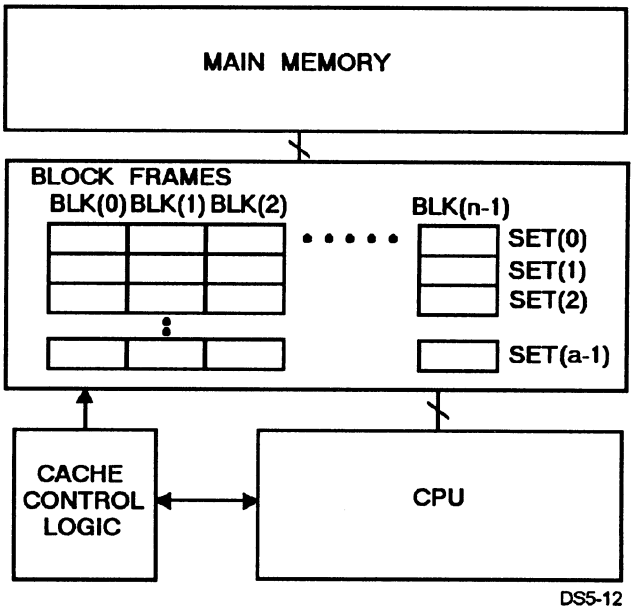

Figure 5-12.—Example of set association mapping used in cache memory.

- Look-through read —In look-through read, the cache is checked first. If a miss occurs, the reference is sent to main memory to be serviced. This is known as a serial read policy.

- Look-aside read —A look-aside read presents both cache and main memory with the reference simultaneously. Since the cache will respond faster, if a hit occurs, the request can be terminated before main memory responds. This is known as a parallel read policy.

CACHE REPLACEMENT POLICIES.— When new data is read into the cache, a replacement policy determines which block of old data should be replaced. The objective of replacement policies is to retain data that is likely to be used in the near future and discard data that won't be used immediately. The replacement policies include the following:

- FIFO —The first block that was read into cache is the first one to be discarded.

- LRU —The block that hasn't been used in the longest period of time is replaced by the new block.

- Random. —Blocks are replaced randomly.

- Optimum— This cache replacement algorithm is psychic and has perfect knowledge of the future. Optimum replacement is what the other three strive for, with LRU coming the closest.

CACHE WRITE.— Since the cache contents area duplicate copy of information in main memory, writing (instructions to enter data) to the cache must eventually be made to the same data in main memory. This is done in two ways as follows:

- **Write-through cache—** Writing is made to the corresponding data in both cache and main memory.

- **Write-back cache—** Main memory is not updated until the cache page is returned to main memory.

READ-ONLY MEMORY (ROM)

Every computer comes with a set of software instructions supplied by the manufacturer. This enables the computer to perform its I/O operations. These permanent instructions (routines) reside in a read-only memory (ROM). ROM is often referred to as **firmware:** software permanently contained in hardware. The instructions are considered permanent or nonvolatile, since they are not erased each time the computer loses power or is turned off. The ROM contains the program that defines its uniqueness compared with all other types of computers.

The ROM is programmed at the time of manufacture and cannot be altered. It is tailored to system requirements. It cannot be altered except by removing and replacing it—either a module or IC chip on a board. The contents of the ROM are electrically unalterable. Other variations of ROMs called PROMS can be reprogrammed as required. This and other variations are covered in further detail in chapter 6 on memory.

In connection with the ROM, you will hear the term *boot procedure* used. The ROM initiates the boot procedure—a sequence of steps followed when you turn on the power to the computer or initiate the boot procedure. The steps required to successfully **boot** the computer depend on the type of computer. Other terms that have the same meaning as boot include **boot up, booting,** or **bootstrap.** They all refer to the process of loading the software. Consult your computer's technical or owner's manual for the exact procedures for your computer system. We use two types of ROMs to discuss some of the programs associated with the ROM: nondestructive readout (NDRO) memory and basic input/output system (BIOS).

Nondestructive Readout (NDRO) Memory

A nondestructive readout (NDRO) memory is usually associated with a militarized mainframe or minicomputer. The NDRO is a small module that occupies two or more slots. For mainframes, it is located in the CPU module. For minicomputers, it is located in the chassis that contains the CPU's pcb's. The functions of an NDRO are controlled from the computer's controlling device: a maintenance console or equivalent. The sizes of the NDRO addresses vary with the type of computer and its requirements. Selection of a particular word in the NDRO is via the NDRO address select, line selector, and current switch logic. AN NDRO consists of hardwired circuits to create the bootstrap programs or a ROM or PROM. Some of the programs contained on an NDRO include the following:

- Two bootstrap programs—Used to load programs from peripheral equipments into main memory

- Autostart programs

- Computer start programs—Used to start a program from a controlling device, locally or remote
- Interrupt routines
- Diagnostic programs—Load failure analysis, memory test, interface test, and computer interconnection system
- Program development memory
- User-specified programs
- Inspect and change programs

Basic InPut/Output System (BIOS)

A basic input/output system (BIOS) is usually associated with "a microcomputer. The BIOS performs the same basic function that an NDRO does in larger computers except for a few major differences. The BIOS is located in the CPU/memory pcb. It is contained on one or more IC chips on the pcb, and the functions of the BIOS are initiated when the computer is powered on. Among the tasks performed are diagnostic testing, environmental inventory, and boot procedure. Figure 5-13 is a basic diagram of installing a BIOS along with the operating system into RAM of a microcomputer.

DIAGNOSTIC TESTING.— Diagnostic testing or Power-on Self Test (POST) is initiated when you initially power up the micro. These tests generally do the following:

- Test CPU registers and flags

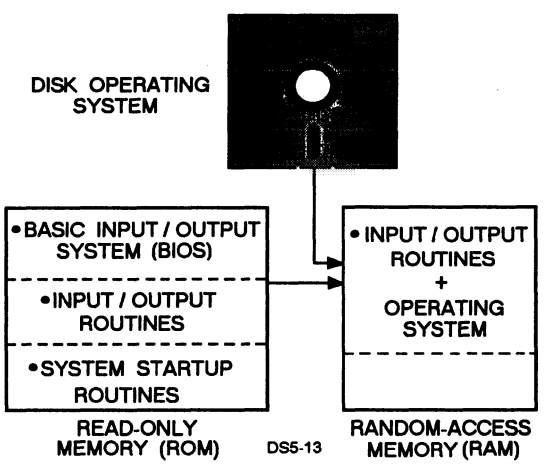

Figure 5-13.—Basic diagram of installing a BIOS.

- Compute and check a checksum for the ROM
- Check the direct memory access (DMA)
- Test the interrupt controller
- Test the timer
- Perform a checksum test on the BASIC (programming language) ROMs
- Test the video
- Test the CRT interface lines
- Test the memory
- Test the keyboard

ENVIRONMENTAL INVENTORY.— This portion of the BIOS includes, just as the name implies, taking inventory of the presence or absence of key items. It includes the following tasks:

- Initialize installed adapters if necessary and return to BIOS startup. Adapters include hard disk controllers, enhanced graphics adapter (EGA), and local-area network (LAN) adapters.
- Check disk controllers for floppy and hard drives.
- Determine the number of printers and serial ports attached.

BOOT PROCEDURE.— Once the testing and inventory are complete, batch files are executed. These are the files that have been written to execute the sequence of instructions needed when the system is powered up and the system configuration files are loaded. The ROM chip program searches for the operating system files on either the floppy drive diskette and/or the hard disk depending on the system setup. As soon as the operating system is located, it is loaded into memory and control is turned over to the operating system. To let you know the microcomputer is ready to use, an opening message (a prompt) is displayed.

TOPIC 2—ARITHMETIC AND LOGIC UNIT (ALU)

The arithmetic and logic unit (ALU), also called the arithmetic section, is designed to perform the arithmetic and logical operations for the CPU. The data required to perform the arithmetic and logical calculations are inputs from the designated CPU registers and operands. The ALU relies on basic items to perform its operations. We have discussed some of these basic items in previous

chapters and topics. They include the number systems, data routing circuits (adders/subtractors), timing, instructions, and operand/registers.

In this topic, we discuss the instructions, timing, and operand/registers and how they apply to the ALU and the ALU operations. Figure 5-14 shows a representative block diagram of an ALU of a microcomputer. Chapter 4 of this volume and NEETS Module 13, *Introduction to Number Systems and Logic Circuits,* provide a review of number systems, adder/subtractor circuits, timing, instructions, and operands/registers. Also refer to NEETS 13 for detailed information of the types of number systems and information basic to all number systems; their identification, operations (addition and subtraction including radix-minus-1 complement and radix-minus-2 complement computations), and conversion. They are discussed in more detail, and it would benefit you to review them to gain a better understanding of how they apply in the ALU operations.

INSTRUCTIONS

The instructions tell the CPU which type of mathematical or logical calculation the ALU will perform. They will also tell the CPU the location of the data on which the ALU will perform the calculations and where to store the results. Results can be used immediately or stored for use later. Special codes within the instructions can also affect arithmetic or logical operations. They can be used for branching or setting flag registers.

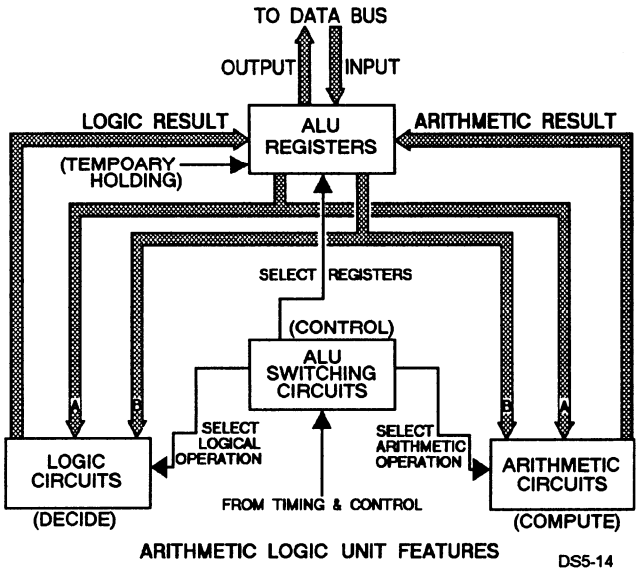

Figure 5-14.—Representative block diagram of an ALU.

TIMING

Timing in the ALU is provided by the CPU's timing circuits. Larger computers have their own arithmetic timing circuits independent of the CPU's timing circuits. In this case, arithmetic timing is initiated by a command from the CPU's main timing chain and the length of the arithmetic timing chain is dependent upon the specific instruction.

OPERANDS/REGISTERS

The registers and operands provide the computer the sources of the data needed to perform the calculations. They also provide the destination for results. Computers can be designed to include the use of **whole-word, half-word,** and **quarter-word operands** and the use of **single-length** and **double-length word/operands** to carry out the arithmetic operations. Double-length memory words or operands will be used for mathematical operations in which the size of the result would be greater than the length of either of the two registers used to provide inputs to the ALU or the operands being input to the ALU are larger than a single word. The sign bit in double-length memory words or operands is the most significant bit (msb). **Flag registers** of one to three bits may be used by the ALU to indicate the status of the last arithmetic or logical operation. The last arithmetic or logical calculation used to set a flag register is often followed by a branching operation. Some of the items indicated by flag registers include the following:

- Equal to zero (= 0)
- Greater than (>)
- Less than (<)
- Positive sign (+)
- Negative sign (–)
- Carry or borrow
- Overflow

Other items used in the ALU include **selectors** and **counters.** The selectors are used to transfer the data between the various registers (accumulators) used in the ALU. Counters are used to keep track of shifts used in the various arithmetic and logical calculations.

ALU OPERATIONS

ALU operations in the CPU include calculations of integers and/or fractions. All the computations are performed using the binary number system. ALU operations also include **signed** arithmetic operations. First we discuss how the binary equivalents of decimal numbers are represented in **fixed-point** representation (integers), then we discuss **floating-point** representation (fractional). Fixed- and floating-point operations are important for the computer. They make the computer versatile when performing arithmetic and logical types of ALU operations.

Fixed-Point Operations

Fixed-point arithmetic operations are performed on integral or whole numbers where the binary point is assumed to be to the right of the least significant bit (bit 0). For example, if we have an 8-bit register, we may express integer decimal numbers between 0 and 2^8 minus 1 (or 255), by converting the decimal number to its binary equivalent. If we have a 16-bit register, we can store integer decimal numbers between 0 and 2^{16} minus 1 (or 65535). Because the binary point is fixed and always to the right of the least significant digit, fractions are not represented. The magnitude or absolute value of the number is always represented by 2^N minus 1 where N is the number of bits within the register or memory cell where the number is being stored.

In fixed-point operations, the computer can perform calculations on signed numbers (positive and negative). The most significant bit (msb) is used as a sign bit. A zero (0) in the msb indicates a **positive** or true form number, and a one (1) in the msb indicates a **negative** or one's complement/radix-minus-1 form number.

When dealing with binary numbers, we can take this one step further; we find the two's complement or radix-minus-2 of the number. It is important to understand the concepts behind 1's and 2's complement. It is the basis by which the computer performs arithmetic and logical calculations. Now if you want to accommodate an equal amount of positive and negative numbers, a 16-bit register can contain numbers from –32768 to +32767 or -2^{15} to 2^{15} minus 1. The reason they are not both 2^{15} is because one combination is taken up for the zero value. This is more easily seen if we examine a 4-bit register. The combinations are shown in table 5-2.

Table 5-2.—Binary and Decimal Values of a 4-Bit Register

(MSB) Bit Position 2^3 (Sign Bit)	Bit Position 2^2	Bit Position 2^1	Bit Position 2^0	Signed Decimal Value
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

That is, there are 2^3 or 2^N combinations and one combination is for the number zero. Negative numbers are represented by their two's complement and the most significant bit (regardless of the word or operand size) is the sign bit. Fixed-point operations can include double-length arithmetic operations, where operands contain 64 bits and bit 2^{63} is the sign bit.

Floating-Point Operations

Floating-point operations are used to simplify the addition, subtraction, multiplication, and division of fractional numbers. They are used when dealing with fractional numbers, such as 5.724 or a very large number and signed fractional numbers. When performing arithmetic operations involving fractions or very large numbers, it is necessary to know the location of the binary (radix) point and to properly align this point before the arithmetic operation. For floating-point operations, the location of the binary point will depend on the format of the computer. All numbers are placed in this format before the arithmetic operation. The fractional portion of the number is called the mantissa and the whole integer portion, indicating the scaled factor or exponent, is called the characteristic.

By rewriting the number in an exponent form, it is often much easier for the computer to manipulate; but, as noted, we give up the digits that were rounded. As a result, some resolution (the number of digits in the fraction) is usually lost. For instance, the number 325786195 could be expressed as 3.26×10^8 or $.32579 \times 10^9$. Still, this concept is useful. The computer, however, is limited by the hardware in the number of bits its registers and memory cells can accommodate.

FLOATING-POINT FORMAT.— The format for the characteristic and mantissa during floating-point operations will vary with the register size. However, the binary (radix) point is usually located between the sign bit and the msb of the mantissa. Typically, floating-point numbers use a 32-bit word size. Let's illustrate a couple of examples—one with a fractional number and another with a very large number. Refer to figure 5-15, frames A and B, during our discussion.

We use one's complement in our examples with 32-bit size words. We'll use the number 6.54321^8 as our example of a fractional number (fig. 5-15, frame A). Our fractional number will require two 32-bit words. In this case, notice the integral characteristic can have a maximum positive or negative value of 2^{15} minus 1 and comprises the least significant 16 bits of the word. Bit 15 contains the one's complement sign, which is extended through the most significant 16 bits of the word. The mantissa is the fractional part of the number and is processed as a 32-bit number including the sign.

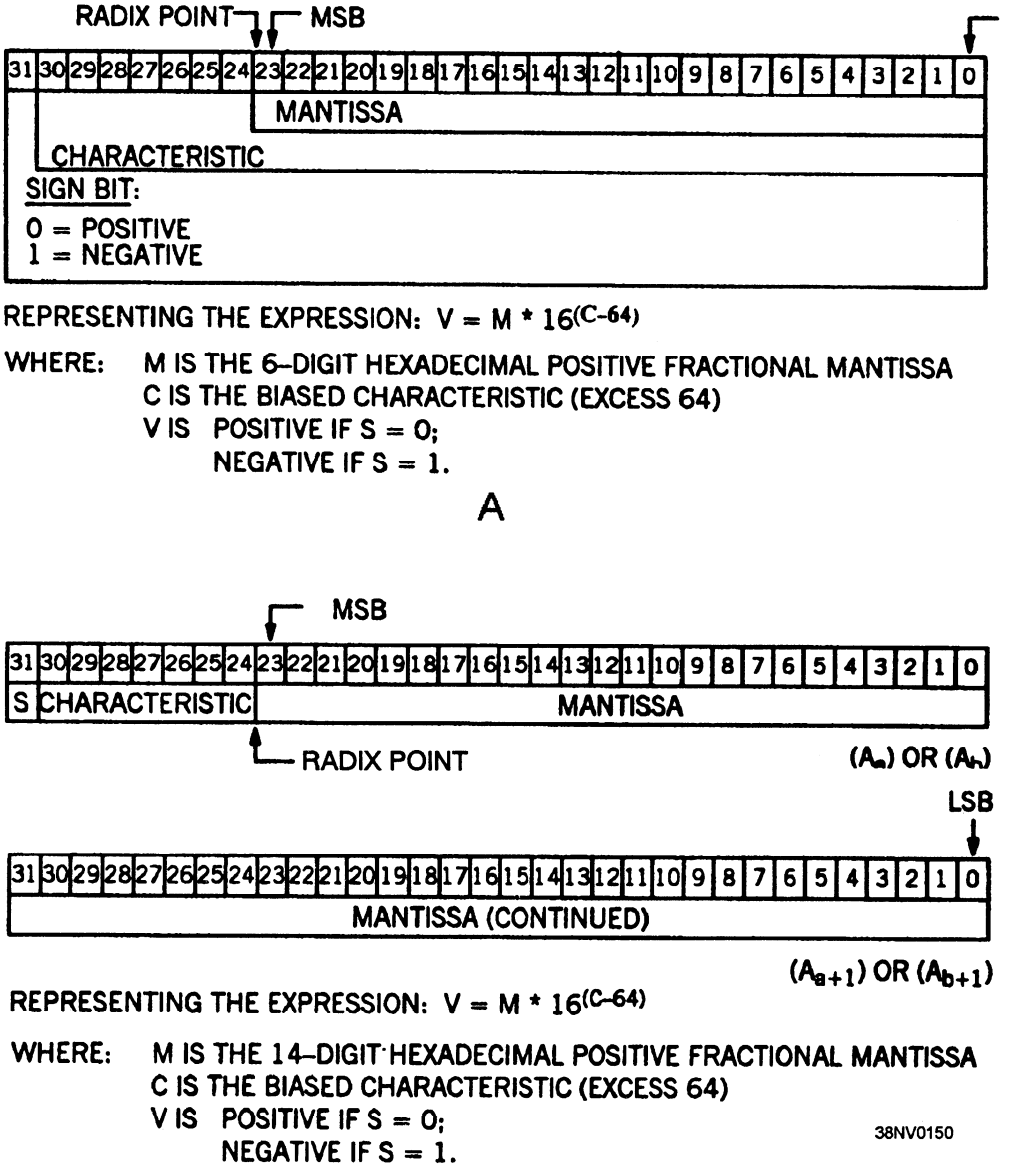

Figure 5-15.—Floating-point numbers: A. Fractional number; B. Very large number.

The second example is a very large number 7665543322211111_8; refer to figure 5-15, frame B. After the number has been put in exponent form, it, too, will require two 32-bit words.

FLOATING-POINT PRECISION.— Floating-point formats include the use of single- and double-precision (refer to figure 5-16, frames A and B). The names single- and double-precision imply their usefulness: **precision.** Notice the double-precision floating-point format, two 32-bit words where the characteristic is small compared to the mantissa in which precision accuracy is required.

FLOATING-POINT ROUND.— Floating-point operations also include **rounding** instructions, which are used for rounding the mantissa's results; rounding up when the mantissa is equal to or greater than one-half of one and rounding down when it less than one-half of one. Rounding can also be applied to double-length results of mantissas. If the sign bit is destroyed (overflowed into) during mantissa rounding or division, the computer will make corrections to the mantissa or quotient.

FLOATING-POINT INTERRUPTS.— Floating-point interrupts can be generated when certain improper conditions are detected. The interrupts inform the program of these conditions and permit either notation or corrective procedures. Some conditions include:

- Underflow (negative excess) or overflow (positive excess)—When a floating-point character exceeds an absolute value of 2^N-1 where N is the msb.
- Divisor —Equals zero in a divide instruction

The control section will be notified and an interrupt will be generated.

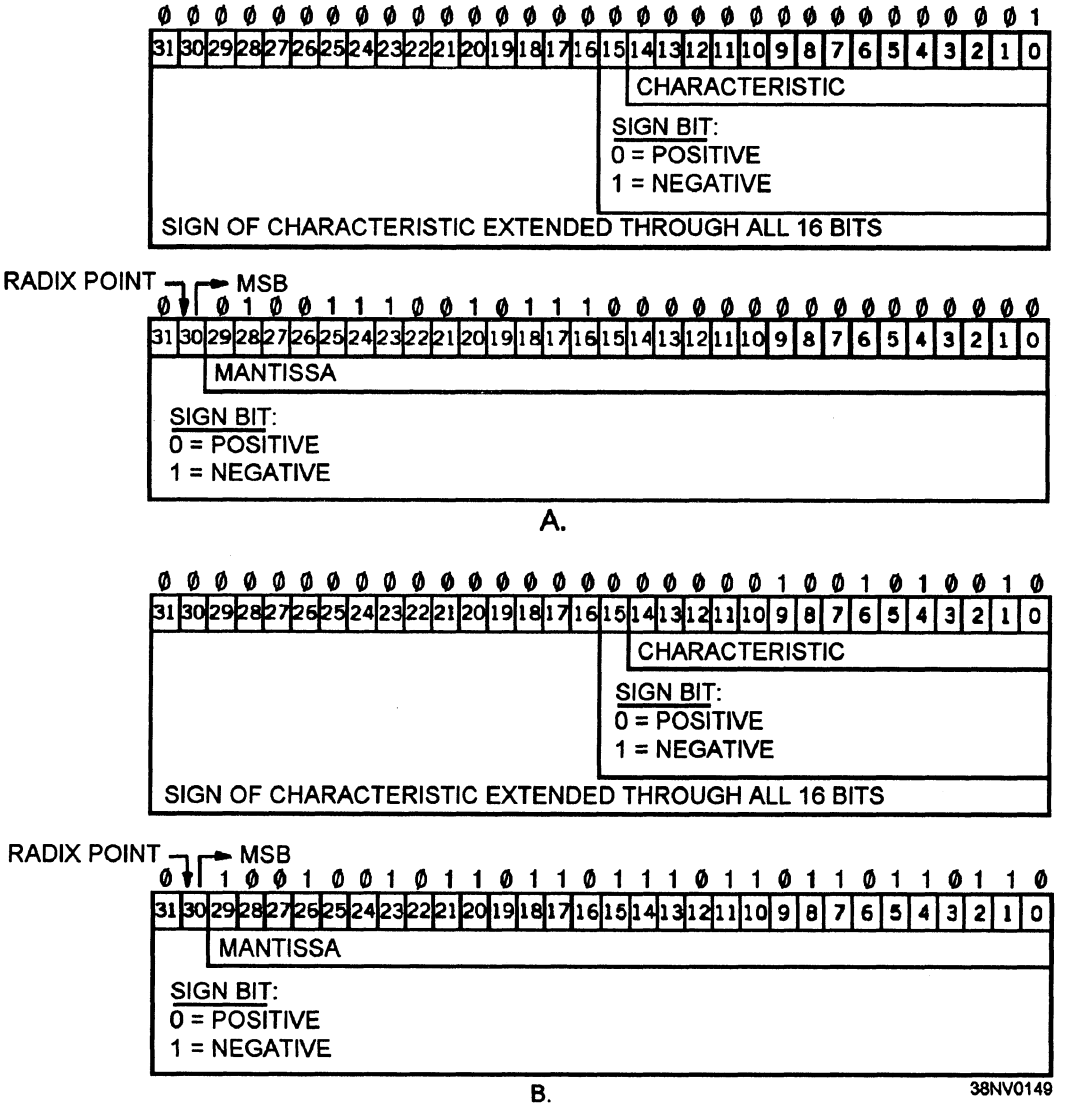

Figure 5-16.—Floating-point numbers: A. Single precision; B. Double-precision.

Operation Types

From the simplest microprocessor (8-bit) to a large mainframe with an embedded microprocessor, the types of ALU operations range from basic add and subtract operations to sophisticated trigonometric operations and separate **coprocessor** and **math pacs,** which operate independent of the ALU. The types of instructions most ALUs can perform can be divided into two categories: **arithmetic operations** and **logical operations.** The ALU uses the logical products of the logic gates to perform the arithmetic and logical instructions. Depending on the sophistication of the computer, the logic gates are arranged to perform the instructions included in the computer's set of instructions.

Computers can be designed to have an adder to perform its adding and subtracting or a subtracter to perform its adding and subtracting. Or they can have a combined adder/subtracter system. Because a computer can really only add or subtract, the add and subtract capabilities allow the computer to perform the more complicated arithmetic operations: multiply, division, and square root functions. Addition and subtraction functions are embedded in division, square root, and the more complicated arithmetic functions, such as trigonometric and hyperbolic, to name a couple.

The computer can be designed where a single instruction will accomplish the results or a series of instructions can be written to produce the results. The only drawback to a series of instruction is they consume more time to accomplish the results. The multiply, divide, square root, and trigonometric instructions are examples.

Computers can multiply by repetitive adding or they can use a series of left shift instructions both using a compare instruction, which may be how a computer with a dedicated multiply function accomplishes the function anyway. The same principle can be applied to the divide and square root functions. A divide can use repetitive subtractions or a series of right shifts with a comparison function. A square root would use a combination of additions/subtractions and comparisons for the multiplying and dividing necessary to accomplish a square root function. A trigonometric function using separate instructions would use logical instructions to accomplish the same results that a single trigonometric instruction would accomplish. ALU operations include signed operations.

Depending on the sophistication of the computer, ALU functions can include the following functions:

- Arithmetic —Add, subtract, shift, multiply, divide, negation, absolute value. (The more sophisticated ALUs can perform square root, trigonometric, hyperbolic, and binary angular movement or motion (B AM) functions.)

- Logical —AND, OR, NOT (complement), and EXCLUSIVE OR (compare).

Also depending on the design, numeric data coprocessor and math pacs are used in some computers in addition to the normal arithmetic instructions available. They execute the arithmetic instructions the CPU's ALU cannot, and they are still controlled by the CPU's program control. These additional logic circuits can be used to amplify the capabilities of the ALU and arithmetic section in general. Remember, the ALU is part of a CPU module or a microprocessor chip on a printed circuit board. The numeric data coprocessor and math pac are separate modules or chips.

NUMERIC DATA COPROCESSOR.— The numeric data coprocessor is a special-purpose programmable microprocessor designed to perform up to 68 additional arithmetic, trigonometric, exponential, and logarithmic instructions. The coprocessor performs numeric applications up to 100 times faster than the CPU alone and provides handling of the following data types: 16-,32-, and 64-bit integers; 32-, 64-, and 80-bit floating-point real numbers; and up to 18-digit binary coded decimal (BCD) operands.

The numeric data coprocessor operates in parallel with and independent of the CPU using the same data, address, and control buses as the CPU. In effect, the coprocessor executes those arithmetic instructions that the CPU's ALU cannot. The CPU is held in a wait mode, while the coprocessor is performing an operation. The CPU still controls overall program execution, while the coprocessor recognizes and executes only its own numeric operations.

MATH PAC.— Math pac is a module used as a hardware option for some militarized minicomputers. The math pac module provides the hardware capability to perform square root, trigonometric and hyperbolic functions; floating-point math; double-precision multiply and divide instructions; and algebraic left and right quadruple shifts.

TOPIC 3—COMPUTER INTERNAL BUSES

To transfer information internally, computers use **buses.** Buses are groups of conductors that connect the

functional areas to one another. This is how the functional areas **communicate** with each other. A bus is a parallel data communication path over which information is transferred a byte or word at a time. The buses contain logic that the CPU controls. The items controlled are the transfer of data, instructions, and commands between the functional areas of the computer: CPU, memory, and I/O The type of information is generally similar on all computers; only the names or terminology of the bus types differs. The name of the bus or its operation usually implies the type of signal it carries or method of operation.

The direction of signal flow for the different buses is indicated on figures in the computer's technical manuals. The direction may be **unidirectional** or **bidirectional** depending on the type of bus and type of computer. Consult the computer's technical manual for details. After becoming familiar with the basic functions and operations of buses, you'll see that regardless of the names, their basic concepts are consistent throughout the computer. They provide avenues for information to be exchanged inside the computer.

BUS TYPES

The preferred method for data/information transfer between system components is by a common data bus. Where point-to-point data transfer is required, the digital format is the preferred method. *General Requirements for Electronic Equipment Specifications, MIL-STD-2036 series,* provides a list of the industry accepted standard internal data buses. They include the standard and the interface as follows:

- IEEE 696—IEEE Standard 696 Interface Devices, S-100
- IEEE 896.l—IEEE Standard Backplane Bus Specification for Multiprocessor Architecture, Future Bus
- IEEE 961—Standard for an 8-bit Microcomputer Bus System, STD Bus
- IEEE 1014—Standard for a Versatile Backplane Bus, VMEbus
- IEEE 1196—Standard for a Simple 32-Bit Backplane Bus, NuBus
- IEEE 1296—Standard for a High-Performance Synchronous 32-Bit Bus, Multibus II

All computers use three types of basic buses. The name of the bus is generally determined by the type of signal it is carrying or the method of operation. We group the buses into three areas as you see them in their most common uses. They are as follows:

- Control (also called timing and control bus), address, and data (also called a memory bus) buses
- Instruction (I), Operand (O), Input/Output Memory (I/O MEM) or Input/Output Controller (IOC), and Computer Interconnection System (CIS)
- Time multiplexed bus

Control Bus

The control bus is used by the CPU to direct and monitor the actions of the other functional areas of the computer. It is used to transmit a variety of individual signals (read, write, interrupt, acknowledge, and so forth) necessary to control and coordinate the operations of the computer. The individual signals transmitted over the control bus and their functions are covered in the appropriate functional area description.

Address Bus

The address bus consists of all the signals necessary to define any of the possible memory address locations within the computer, or for modular memories any of the possible memory address locations within a module. An address is defined as a label, symbol, or other set of characters used to designate a location or register where information is stored. Before data or instructions can be written into or read from memory by the CPU or I/O sections, an address must be transmitted to memory over the address bus.

Data Bus

The bidirectional data bus, sometimes called the memory bus, handles the transfer of all **data** and **instructions** between functional areas of the computer. The bidirectional data bus can only transmit in one direction at a time. The data bus is used to transfer instructions from memory to the CPU for execution. It carries data (operands) to and from the CPU and memory as required by instruction translation. The data bus is also used to transfer data between memory and the I/O section during input/output operations. The information on the data bus is either written into

memory at the address defined by the address bus or consists of data read from the memory address specified by the address bus.

Figure 5-17 is an example of a computer's bus system; control, address, and data buses.

Instruction (I) Bus

The instruction (I) bus allows communication between the CPU and memory. It carries to the CPU the program instruction words to be operated on by the CPU from memory or returns instructions to memory. The I bus is controlled by the CPU. It is capable of sending or receiving data while the operand(O) bus is receiving or sending data at the same time, but only in one direction at a time.

Operand (O) Bus

The operand (O) bus allows communication between the CPU and memory or the CPU and an I/O Controller (IOC). The CPU controls the operation in both cases. The O bus is capable of sending or receiving data, while the I bus is receiving or sending data at the same time, but only in one direction at a time. The direction of the data depends on whether the CPU is reading data from memory or data is being written back into memory.

I/O MEM Bus or Input/Output Controller (IOC) Bus

The I/O memory bus allows communication between an I/O controller (IOC) and memory. It is controlled by the IOC. To respond to the CPU, the I/O MEM bus must use the O bus.

Figure 5-18 is an illustration of communications between a CPU, memory, and an IOC without a computer interconnection system. Pay close attention to the direction of signal flow and which buses allow communication between functional areas.

Computer Interconnection System

The Computer Interconnection System (CIS) provides the complete functional replication of the computer **intraconnection** among CPUs, IOCs, and memories in separate computers. This allows the internal buses to be extended beyond their own enclosure. The CIS consists of two independent halves: the requestor extension interface (REI) and the direct memory interface (DMI).

REQUESTOR EXTENSION INTERFACE (REI).— The requestor extension interface (REI) is a bus extender. It extends the bus up to 15 other computer cabinets providing an interconnected system of memory modules, CPUs, and IOCs. The REI takes the requests from the requestor ports and goes through a priority network to determine the order in which it is to respond to the requestors. Once the REI has responded to a request, it puts the address onto the output bus,

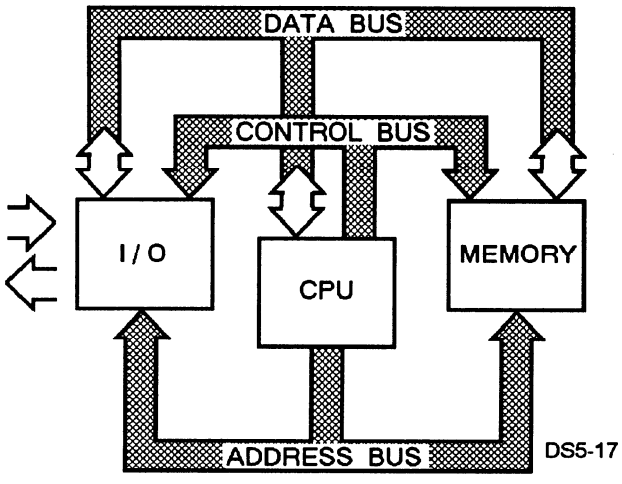

Figure 5-17.—Example of a computer's bus system; control, address, and data buses.

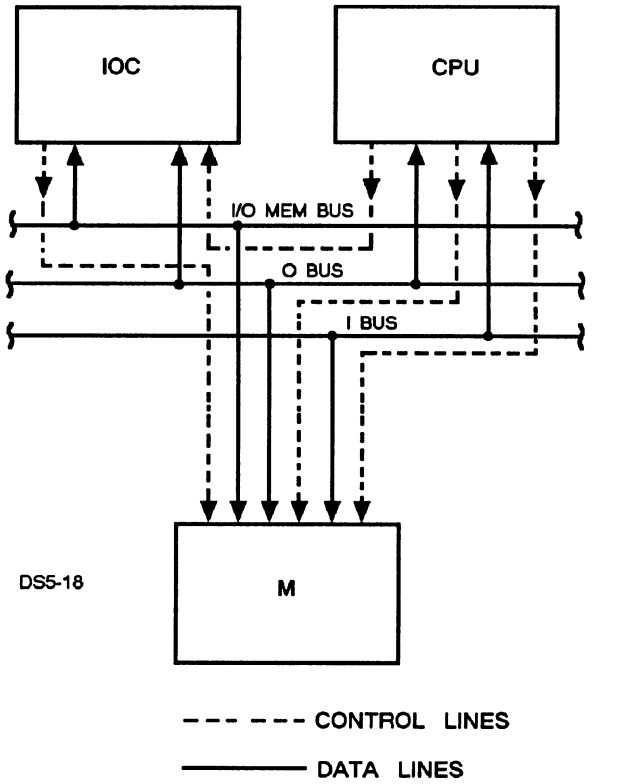

Figure 5-18.—Bus system between a CPU, memory, and IOC without CIS.

checks parity, and examines a code to determine the correct sequence. After the sequence is established, the REI broadcasts the requests and the address to all DMIs connected to it. The signals on the REI external interface are expanded to guarantee capture at the DMI operating synchronously to the REI, which can be located up to 500 cable-feet away. Once the REI makes a request, it can send write data if it is performing a write operation or wait for a response and pass it to the requestor. The REI responds to the requestor just as memory does, including faults and aborts (terminates a process before it is completed).

DIRECT MEMORY INTERFACE (DMI) BUS.— The Direct memory interface is a responder or slave on the REI bus. The DMI bus is used in some computers that use an I, O, and IOC bus. The DMI bus is used to send requests from other enclosures (computers) to the module (CPU or IOC) requested. It acts as the requestor and makes requests to the CPU. When it requests an IOC, it uses IOC read and write requests. When it requests memory, it uses operand read or write, instruction read, or replace.

Time Multiplexed Bus

Another variation of the address and data bus is the time multiplexed bus. This single bus transmits both addresses and data using a four cycle clock (t1, t2, t3, and t4). The address is transmitted during the t1 clock cycle, the direction of data movement is selected during t2, and the data is transmitted during t3 and t4.

BUS OPERATIONS

The bus control function is performed by a bus interface unit or logic circuitry similar to it. Control of a bus line and the proper protocol of requesting a bus depends on the design of the computer. In computers with no IOC, the CPU has control of the bus lines. In computers with an IOC, the CPU will control the instruction and operand buses and the IOC will control the memory buses. Bus control is necessary to handle the large number of bus transactions that take place in a very short period of time in the computer. There are basically two factors that must be taken into consideration in bus communications: **transfer priority** and **source/destination** of the data being transferred.

Bus transfers are done on a priority basis. The priorities of bus transfers are determined by the design of the computer's firmware. What part makes the request is also determined by the design of the computer's firmware; requests may be made by a CPU, an IOC, and/or a DMI. Examples of priorities that a computer must deal with include the following (these examples are not in any type of priority and do not cover the full range of priorities you may encounter):

- Transfers from memory to the CPU, these transfers move instructions and operands to the CPU for execution and modification
- Transfers from the CPU to memory
- Transfers by the I/O in and out of memory

The specific request will identify the source and the destination of the data. The computer's controlling bus continually and repeatedly checks the bus signal lines for requests. When it receives a request, it provides the control signals needed to initiate the transfer. Since most transfers deal with memory, each transfer consists of an address exchange and a separate data exchange. The data will either parallel the address as in a write operation or move in the opposite direction after the data has been read from the memory word identified by the address.

In some computers, the bus systems use **holding registers** in both the source and destination sections to prevent data loss and to help coordinate the data exchange. In the source logic, the data is placed in a holding register until it is accepted by the destination logic. The outputs of the holding register feed the bus circuitry. In the destination logic, the bus inputs to a holding register. After accepting the data, the destination logic can then move the data from the holding register to other parts of the logic for processing.

A variety of command signal names are used to coordinate the exchange of data on the buses by both the source and the destination logic. The source logic generates a ready or signal equivalent when the data is in the holding register and on the bus. The destination logic sends an accept or equivalent signal when it has sensed the ready signal and captured the data on the bus in its holding register or other logic circuits.

MICROCOMPUTER ARCHITECTURE AND BUSES

The microcomputer has uses four main types of buses. These are the

- Processor bus
- Address bus

- Memory bus
- I/O bus

The I/O bus has historically been the slowest of all buses, and the main focus when computer design engineers try to improve bus speeds.

Processor Bus

The processor bus is communications path between the CPU and the main bus. It is also used for communications between the CPU and the processor support chipset. The processor support chipset includes chips such as an external memory cache and the bus controller chip found on some microcomputers. The size of the processor bus matches the size of the data words used by CPU. For example, the 80486DX chip uses 32-bit words; therefore the processor bus has 32 data lines, 32 address lines, and the control lines. The Pentium processors have 64-bit words and use 32-bit addresses. Processor buses can have a maximum data transfer rate of the motherboard clock.

Memory Bus

The memory bus transfers data between the RAM and the CPU. This bus can be the processor bus or will be implemented by a dedicated chipset that controls the memory bus. In most computers that have a motherboard clock that is faster then 16MHz, a special memory controller chipset will control the memory bus.

Address Bus

The address bus transfers the next memory or I/O address to be used in the next data transfer. The address bus in 486 and Pentium systems is 32 bits wide.

I/O Buses

To thoroughly understand the I/O buses used in modern microcomputer systems, an understanding of the development and evolution of bus systems is required. The microcomputer's architecture is directly related to the type of buses in the computer. Originally, microcomputers used a bus system called the S-100 bus. Using this system, any board could be plugged into any open slot. The S-100 bus has 62 lines, each connect to each of the 62-pin connectors. This system dedicated eight lines for the eight data bits used in the Intel 8088 microprocessor. Twenty lines are used for memory addressing. The same 20 lines are also used to address I/O devices. A control line determines whether the data on these 20 lines will be a memory address or an I/O address. There are also several control lines and power distribution lines.

The S-100 bus also provided four lines to designate channels for Direct Memory Accessing (DMA). A DMA channel allows a device, such as the hard drive, to transfer data directly into RAM, vice transferring data to the CPU and then having the CPU transfer it to the RAM. The DMA channel number identifies which device is requesting and transferring data on the data bus.

Buses also need to be clocked to properly transfer data. The early microcomputer buses were designed to run at the speed of the microprocessor that was installed on the board. The 4.7 MHZ 8088 microprocessor clock was also used to clock the bus. The 7.16 MHZ microprocessor clocked the bus at the same rate. The ISA standard set the bus clock speed at 8 MHZ. To maintain compatibility with the older controller boards, this speed is still common in many computers today. This speed is fine when getting input from a mouse or a keyboard, even for most disk drives. The biggest problem with bus speeds has occurred because of the increase in video resolution, the development of video capture boards and some network interfaces.

INDUSTRY STANDARD ARCHITECTURE (ISA).— As the microcomputer evolved, the eight data lines and 20 address lines became insufficient to handle the increased data capacity of the 16-bit processor. This led to the development of the Industry Standard Architecture (ISA). To be compatible with the boards used in eight-bit computers, an additional 36-wire connector was added to the circuit boards and the bus. This added eight more data lines, four more address lines, four more DMA channels, and five more IRQ channels.

LOCAL BUSES.— A local bus is a bus that is a dedicated path between the processor and a specific board. There are several local buses built into various types of computers to increase the speed of data transfers. Local buses for expanded memory and video boards are the most common. Some high-end computers also provide a local bus for the hard drive.

The VESA Local Bus is one of the more popular buses and was developed to increase the speed of data transfer between memory and the video processing board (video graphics adapter). VESA stands for Video Electronics Standards Association. The VESA Local Bus is a direct bus that connects the video processor

with the processor bus. The VESA Local Bus operates at the speed of the video processor.

Several other bus systems have been developed, many of which have not found widespread acceptance in the PC world. Each of these has introduced some technology that is common in the modem bus systems.

MICROCHANNEL ARCHITECTURE (MCA).— The MicroChannel Architecture (MCA) bus was developed by IBM in 1987 and increased the bus speed to 10 MHZ. The MCA Bus also introduced the ability to configure the boards IRQ and DMA channels through a software configuration program. MCA was the first system to use bus mastering. Bus mastering is a system that allows an intelligent controller board to take control of the bus system for a specified period of time. This allows operations to be completed quickly. Bus mastering differs from DMA in that DMA allows for direct transfer from a peripheral controller to RAM, Bus mastering allows for direct transfers between controllers. An example of bus mastering is the ability of a hard drive to transfer graphics directly to the graphics driver, bypassing the CPU and RAM.

The major disadvantage of MCA was that it is not compatible with the old ISA standard. Therefore, if you have an MCA machine, the old ISA controller boards will not work.

EXTENDED INDUSTRY STANDARD ARCHITECTURE.— To compete with MCA, The Extended Industry Standard Architecture was (EISA) developed. The EISA Bus included the following features:

- 32-bit data path
- 64K of I/O address
- Capability to address up to 4 giga-bytes of memory
- Software configuration of boards
- Bus mastering

Unfortunately, the EISA Bus still operates with an 8 MHZ clock, and did not add any additional DMA or interrupt channels.

PERIPHERAL COMPONENT INTERCONNECT (PCI).— The Peripheral Component Interconnect (PCI) system was designed to increase I/O bus speeds while still maintaining compatibility with previous ISA and EISA boards. A PCI computer has two separate banks of expansion slots, one bank for PCI boards and one bank for the older ISA/EISA boards. The PCI bus uses a "bridge circuit" to isolate the processor bus from the main I/O bus. This bridge circuit is designed so that I/O functions can run independently from the CPU.

The PCI bus is a 64-bit data bus, but can also support 32-bit computers. This makes the PCI bus useful in both Pentium and 486 systems. The PCI bus can operate a speed up 33 MHZ and also supports bus mastering. Finally, the PCI bus supports the Plug-n-Play standard for software configuration of peripheral boards.

SUMMARY—CENTRAL PROCESSING UNITS AND BUSES

This chapter has introduced you to central processing units (CPUs) and buses. The following information summarizes important points you should have learned:

CENTRAL PROCESSING UNITS— All the computational operations (logical and arithmetic) and operational decisions are made in the CPU. The CPU controls all computer operations. The CPU has a control section and an arithmetic logic unit (ALU).

CONTROL SECTION— The control section directs the sequence of CPU operations, interprets the instructions, and provides the timing and control signals to carry out the instructions.

TIMING— Thing in a computer regulates the flow of signals that control the operation of the computer. Computer operations rely on both synchronous and asynchronous operations. Timing circuits are used throughout the computer.

INSTRUCTION AND CONTROL— The instruction execution and control portion of the control section includes the combinational and sequential circuits that make up the decision-making and the memory-type functions. The general process of execution of a machine instruction is fetch the instruction, update the program counter or equivalent, translate the instruction, and execute the instruction.

INTERRUPTS— Interrupts are a method of diverting the attention of the computer from whatever process or program it is performing to handle the special condition or event that caused the interrupt signal. Interrupts allow the computer to respond to high priority demands and still be able to perform normal or lower priority processing. An interrupt is defined as a break in the normal flow of operation of a computer caused by an **interrupt signal.** The break occurs in

such a way that the operation can be resumed from the point of the break at a later time with exactly the same conditions prevailing. CPUs follow a specific sequence of events when processing an interrupt. Interrupt processing has priority over normal program execution.

CONTROL MEMORY— Control memory consists of addressable storage registers. It is used as a temporary storage. Access to control memory data requires less time than access to main memory. This speeds up CPU operation by reducing the number of memory references for data storage and retrieval.

CACHE MEMORY— Cache memory is a small, high-speed RAM buffer located between the CPU and main memory and used to hold a copy of the instructions or data currently being used by the CPU. It is used to speed up the flow of instructions and data into the CPU from main memory.

READ-ONLY MEMORY— Every computer is supplied with a set of software instructions to enable the computer to perform its I/O operations. These permanent instructions (routines) reside in a read-only memory (ROM). ROM is often referred to as firmware: software permanently contained in hardware. The instructions are considered permanent or nonvolatile, since they are not erased each time the computer loses power or is turned off. The ROM is tailored to system requirements and initiates the boot procedure —the steps followed when you turn on computer power.

ARITHMETIC LOGIC UNIT— The arithmetic logic unit (ALU) implements arithmetic and/or logical operations required by the instructions. The instructions tell the CPU which type of mathematical or logical calculation the ALU is to carry out. The registers and operands provide the computer the sources of the data needed to perform the calculations. Timing in the ALU is provided by the CPU's timing circuits.

ALU OPERATIONS— ALUs can perform arithmetic and logical operations. An ALU can be designed to perform arithmetic operations in fixed-point representation (integers) and floating-point representation (fractional). The types of arithmetic operations range from add and subtract operations to sophisticated trigonometric operations. Some computers have a separate numeric data coprocessor or math pacs to perform arithmetic functions independent of the ALU.

INTERNAL BUSES— Buses transfer information internally in computers. A bus is a parallel data communication path over which information is transferred a byte or word at a time. The direction of signal flow may be unidirectional or bidirectional.

BUS OPERATIONS— The bus control function is performed by a bus interface unit or logic circuitry similar to it. Control of a bus line and the proper protocol of requesting a bus depend on the design of the computer. Bus transfers are done on a priority basis. Basically two factors must be taken into consideration in bus communications: transfer priority and source/destination of the data being transferred.

By studying this chapter, you should have learned how the CPU works through its control section and its arithmetic logic unit. You also should have learned how buses are used to transfer instructions, data, and information throughout a computer. These concepts are important to understanding how to troubleshoot and diagnose malfunctions and repair or replace CPU parts.